School of Computation, Information and Technology
Technische Universität München

Lehrstuhl für
Rechnerarchitektur & Parallele Systeme
Prof. Dr. Martin Schulz
Dominic Prinz
Jakob Schäffeler

Lehrstuhl für
Design Automation
Prof. Dr.-Ing. Robert Wille
Stefan Engels

Einführung in die Rechnerarchitektur

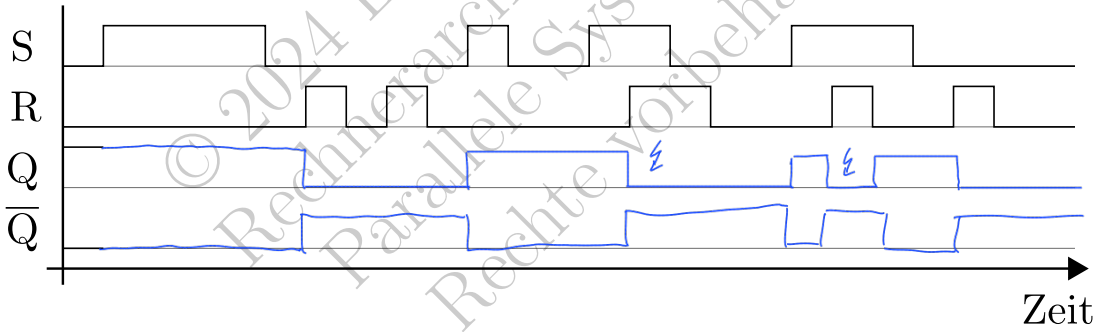
Wintersemester 2024/2025

Übungsblatt 7: Sequenzielle Logik 02.12.2024 – 06.12.2024

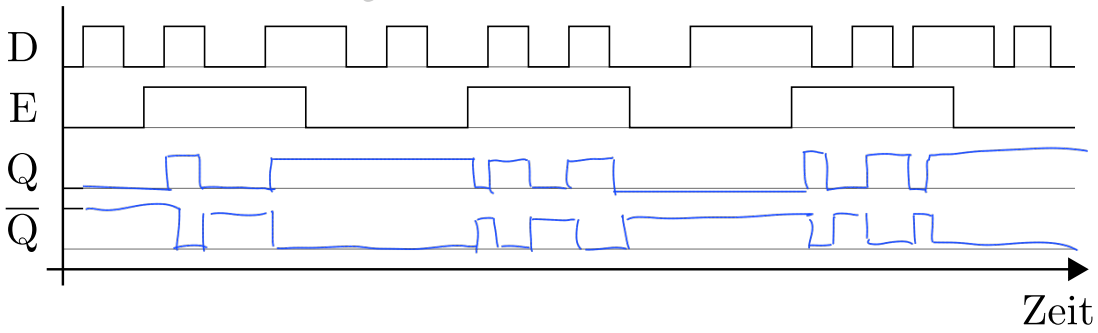
1 Wellenformen

Vervollständige die Wellenformen für folgende Speicherbausteine.

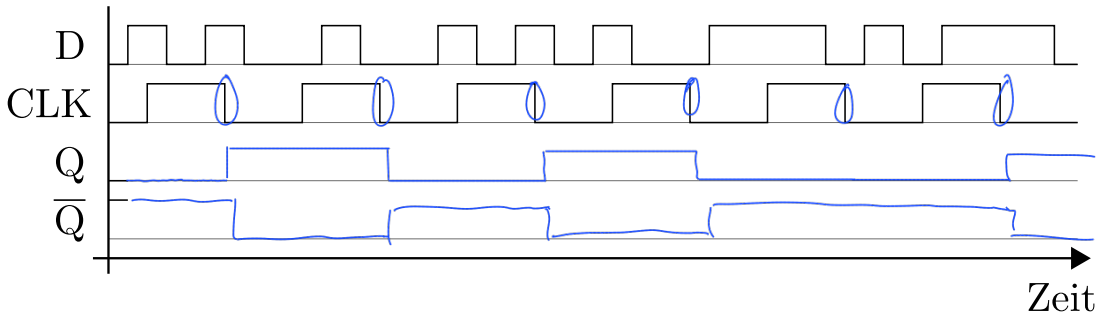
a) RS-Flipflop



b) D-Latch



c) Taktflankengesteuertes D-Flipflop (fallende Flanke)



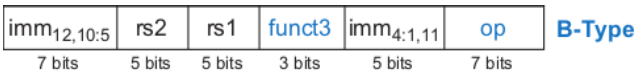
2 Program Counter

Der *Program Counter* (PC) ist ein Zähler im Prozessor der die Addresse des nächsten auszuführenden Befehls speichert. Der PC wird in jedem Befehlszyklus erhöht, sodass er die Adresse des *nächsten* Befehls im Befehlsspeicher berechnet. Da RISC-V Instruktionen 32-bit lang sind, wird der PC jedes mal um 4 erhöht (32 Bit = 4 Byte).

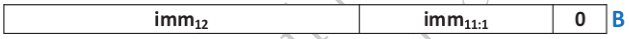
1. Implementiere die Grundfunktionalität des Program Counters in Digital. Bei steigender Taktflanke soll der PC um 4 erhöht werden. Weiterhin soll es möglich sein den PC mittels Resetsignal auf 0 zu setzen.

2. Um interessante Programme ausführen zu können, muss es möglich sein den PC auch „manuell“ zu schreiben. In RISC-V wird das über *Jump-* und *Branchbefehle* erreicht. Die Logik des PCs soll erweitert werden um auch den `BEQ` (branch if equal) Befehl des RISC-V Assembly umzusetzen.

Der `BEQ` befehl vergleicht zwei Register und addiert einen relativen Offset zum aktuellen PC falls diese gleich sind. Der relative Sprung ist in dem 12-bit *Immediate* Feld des Maschinencodes gespeichert. Da laut RISC-V Standard alle Instruktionen 2-Byte-Aligned[1] sind, wird das niederwertigste Bit (welches immer 0 ist) nicht gespeichert und nur implizit behandelt. Tatsächlich werden durch das 12-bit Immediate also implizit 13-bit dargestellt. Der Maschinencode des `BEQ` ist wie folgt definiert:



Die relative Sprungadresse ist dabei in den `imm` Bits codiert. Die 32-bit Konstante die dadurch spezifiziert ist, berechnet sich wie folgt:



(1)

Das *Most Significant Bit* des Immediates definiert also das Vorzeichen. *Sign Extension* ist eine Berechnung die häufig in Prozessoren vorkommt wenn Signale mit wenigen Bits auf Wortlänge erweitert werden sollen.

Implementiere zunächst eine Komponente `Extend` in Digital die eine 12-bit Zahl wie in obiger Abbildung (1) auf 32-bit erweitert. Wichtig hier ist, dass nur das unterste Bit des erweiterten Vectors auf ’0’ ist. Daher wird der PC immer um den doppelten Wert des offsets verändert.

[1] In der Vorlesung werden nur 4-Byte Instruktionen behandelt, es gibt jedoch Erweiterungen, die auch compressed 2-Byte Instruktionen zulassen.

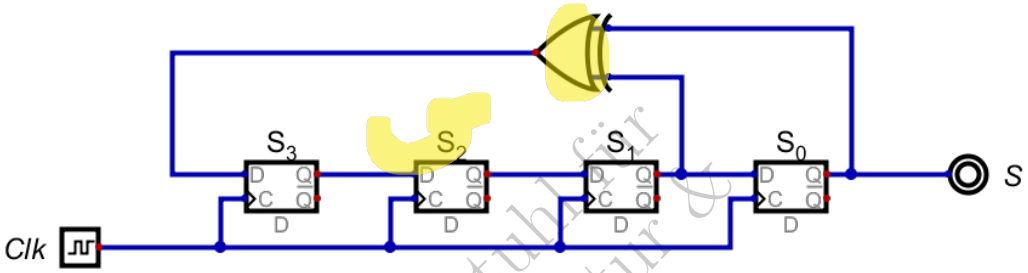
3. Erweitere nun die Logik des PC sodass der PC um eine 12-bit relative Adresse `offset` erhöht bzw. verringert werden kann. Ein Signal `pcsrc` bestimmt dabei ob der PC normal (um 4 erhöht) oder mit relativer Adresse berechnet werden soll. Verwende hierfür die Komponente aus dem vorigen Beispiel.

Überprüfe die Richtigkeit deines Designs mittels Simulation in Digital.

3 Linear-Rückgekoppeltes-Schieberegister

Ein Schieberegister kann zur Realisierung einer First-In-First-Out Warteschleife (kurz FIFO) in digitaler Hardware verwendet werden. Neben der Verwendung eines Schieberegisters als FIFO-Speicher kann das Schieberegister auch zur Erzeugung von periodischen, deterministischen Zufallszahlen verwendet werden. Dazu wird ein sogenanntes Linear-Rückgekoppeltes-Schieberegister (Linear-Feedback-Shift-Register; kurz: LFSR) verwendet. In diesem Beispiel soll nun das LFSR in der nachfolgenden Abbildung analysiert werden.

Dabei sollen die Zustände der D-FlipFlops $S_3, \cdots, S_0$ über die gesamte Periode angegeben werden, sowie zusätzlich die generierten Zufallsbits S. Es soll davon ausgegangen werden, dass diese initial im Zustand $(S_3\ S_2\ S_1\ S_0)_2 = (0\ 1\ 0\ 0)$ sind.



S_3	S_2	S_1	S_0	$XOR(S_1, S_0)$	S
0	1	0	0	0	0
0	0	1	0	1	0
1	0	0	1	1	1
1	1	0	0	0	0

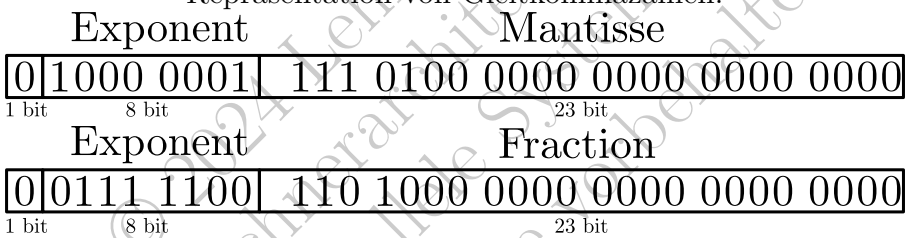
4 Addition von Gleitkommazahlen (Hausaufgabe)

Bearbeitung und Abgabe der Hausaufgabe 7 auf `https://artemis.in.tum.de/courses/401` bis **Sonntag, den 08.12.2024, 23:59 Uhr**.

Neben den bereits bekannten arithmetischen Schaltungen für Ganzzahlrepräsentationen wird in dieser Übung ein Single-Precision (32-Bit) Addierer für Gleitkommazahlen untersucht. Das Gleitkommazahlensystem bietet gegenüber der festen Darstellung von Ganzzahlen den großen Vorteil, dass sowohl sehr große als auch sehr kleine Zahlen dargestellt werden können. Ähnlich zur wissenschaftlichen Notation von Zahlen besitzen Gleitkommazahlen ein Vorzeichen (*sign*), eine Mantisse (*mantissa*), eine Basis (*base*) und einen Exponenten (*exponent*). Die Zahl 5300 hat beispielsweise in wissenschaftlicher Schreibweise die Darstellung 5.3×10^3 mit der Mantisse 5.3, der Basis 10 und dem Exponenten 3. Der Dezimalpunkt wird dabei so verschoben, dass er hinter das höchstwertige Bit (*MSB*) gesetzt wird. Gleitkommazahlen verwenden die Basis 2 und eine binäre Mantisse. Die 32 Bit sind aufgeteilt in 1 Bit für das Vorzeichen, 8 Bits für den Exponenten und 23 Bits für die Mantisse.

Im Floating-Point-Format ist das erste Bit der Mantisse immer 1 und muss daher nicht gespeichert werden. Dieses System wird als *implicit leading one* bezeichnet. Dadurch werden nur die *Fraction*-Bits gespeichert, und ein weiteres Bit steht zur Verfügung.

Repräsentation von Gleitkommazahlen:



Um sowohl positive als auch negative Exponenten darzustellen, wird ein *biased* Exponent verwendet. Bei Single-Precision wird ein Bias von 127 verwendet. Für einen Exponenten von -4 ergibt sich somit ein biased Exponent von $123 = 01111011_2$.

Die Addition von Gleitkommazahlen umfasst die folgenden Schritte:

1) Extraktion von Exponent und Fraction
2) Ergänzung der führenden 1 zum Erhalt der Mantisse
3) Vergleich der Exponenten
4) Shift der kleineren Mantisse (falls notwendig)
5) Addition der Mantissen
6) Normalisierung der Mantissen (falls notwendig)
7) Rundung des Ergebnisses
8) Zusammenfügen von Exponent und Fraction in eine Gleitkommazahl

Ein Beispiel findet sich in Abbildung 1 am Ende dieses Übungsblattes.

Von folgenden Vereinfachungen darf ausgegangen werden:

- Es werden ausschließlich positive Zahlen addiert, d.h., das Vorzeichen-Bit ist immer 0.
- Falls ein Rundung notwendig wird, so wird in Richtung Null gerundet (truncate).
- Die Sonderfälle 0, $\pm\infty$, NaN (Not a Number) werden ignoriert.

Die Schritte sollen einzeln in Digital implementiert werden und dann zu einem floating-point Adder zusammengefügt werden:

a) **Extraktion von Mantisse und Exponent**: In dieser Teilaufgabe soll eine Schaltung für Schritt 1 und 2 entworfen werden. Da beide Zahlen positiv sind, kann das Vorzeichen ignoriert werden. Die nächsten 8 Bits bilden den Exponenten, und die verbleibenden 23 Bits bilden die Fraction. Für die Ausgabe der Mantisse muss schließlich noch die führende 1 ergänzt werden.

   Die Komponente zur Extraktion hat also einen 32-bit Input (`Float`) und einen 8-Bit-Exponenten (`Exp`) sowie eine 24-Bit-Mantisse (`Mant`) als Output.

   Vervollständige die Schaltung in `1_ExtractBits.dig`.

b) **Vergleich der Exponenten**: In dieser Teilaufgabe soll eine Schaltung für Schritt 3 entworfen werden. Das Ergebnis bezieht sich am Ende auf den größeren Exponenten. Die kleinere Zahl muss daher an die größere Zahl angepasst werden. Zunächst muss die Differenz der Exponenten berechnet werden.

   Die Eingänge dieser Schaltung sind die beiden Exponenten `ExpA` und `ExpB`, jeweils 8-bit. Die Ausgänge sind wie folgt definiert:

   - `shamt` (8bit): Die *positive* Differenz der Exponenten, d.h. $|ExpA - ExpB|$
   - `ExpA<ExpB` (1bit): Dieser Ausgang soll 1 sein genau dann wenn $ExpA < ExpB$
   - `Exp` (8bit): Der unveränderte *größere* Exponent, d.h. `ExpB` wenn $ExpA < ExpB$ und `ExpA` andernfalls

   Vervollständige die Schaltung in `2_ExponentCompare.dig`.

   *Hinweis: Berrechne die beiden Differenzen $ExpA - ExpB$ und $ExpB - ExpA$ separat. Das carry out von $ExpA - ExpB$ kann als Output eines Komparators verstanden werden und erzeugt **`ExpA<ExpB`**. Gleichzeitig kann dieses Signal entscheiden, welche Differenz und welcher originale Exponent zu den jeweiligen outputs geleitet werden sollen.*

c) **Shift der Mantisse basierend auf der Exponentendifferenz**: In dieser Teilaufgabe soll eine Schaltung für Schritt 4 entworfen werden.

   Die Schaltung hat das Signal `ExpA<ExpB` (1bit), die Differenz der Exponenten `shamt` (8bit) und die Mantissen `MantA` und `MantB` (je 24 bit) als Input. Als Ausgang `ShiftedMant` (24bit) soll die kleinere der beiden Mantissen um `shamt` nach rechts geshifted ausgegeben werden.

Vervollständige die Schaltung in `3_ShiftMantissa.dig`.

Hinweis: Das Signal ***`ExpA<ExpB`*** *entscheidet, welche Mantisse (A oder B) als Input genutzt wird und entsprechend nach rechts geshifted wird. Zusätzlich gilt: Wenn die Bits* 7*,* 6*,* 5 *oder* (4 *und* 3) *von shamt auf* 1 *gesetzt sind, kann die geshiftete Mantisse auf* 0 *(alle* 24 *Bits) gesetzt werden, da die Zahl dann als zu klein betrachtet wird, um einen Unterschied in der Summe darzustellen.*

d) **Addition der Mantissen**: In dieser Teilaufgabe soll eine Schaltung für Schritte 5 bis 7 der Mantisse entworfen werden. Dafür werden die Mantissen aufaddiert und falls das Ergebnis ein carry out hat muss das Ergebnis eins nach rechts geshiftet werden und der Exponent um eins erhöht werden.

Zuerst betrachten wir nur die Addition der Mantissen und den möglichen shift. Die Schaltung hat als Input wieder das Signal `ExpA<ExpB`, die zwei originalen Mantissen `MantA` und `MantB` und die geshiftete Mantisse `ShiftedMant`. Die Ausgänge sind wie folgt definiert:

- `Fract` (23bit): Fraction (Mantisse ohne die führende Eins) des Ergebnisses
- `ShiftExp` (1bit): Zeigt an, ob der Exponent geshifted werden muss, da es bei der Addition zu einem carry-out kam.

Vervollständige die Schaltung in `4_AddMantissaAndNormalize.dig`.

Hinweis: ***`ExpA<ExpB`*** *entscheidet wieder, welche der Mantissen im Original verwendet wird. Diese Mantisse wird mit der geshifteten Mantisse addiert. Das Ergebnis hat nun* 24 *Bits. Falls das Carry-Out-Bit der Addition* 1 *ist, werden die Bits* [23 : 1] *verwendet; andernfalls werden die Bits* [22 : 0] *als Fraction-Output ausgegeben. Implizit wird dadurch bereits die führende 1 gestrichen und gegebenenfalls (im Falle eines Carry-Out-Bits) normalisiert und in Richtung 0 gerundet (truncate). Das Carry-Out muss ebenfalls als Output ausgegeben werden, da es im nächsten Schritt zum Exponenten addiert werden muss.*

e) **Anpassung des Exponenten**: In dieser Teilaufgabe soll eine Schaltung für den vebleibenden Teil von Schritt 6 (Exponent) entworfen werden. Falls im vorherigen Schritt ein Carry-Out-Bit auftrat (Input `AddExp`, 1bit), so muss der Exponent (Input `Exp0`, 8bit) um eins erhöht werden und wird als `Exp` (8bit) ausgegeben.

Vervollständige die Schaltung in `5_AddExp.dig`.

f) **Zusammenfügen zu einer Gleitkommazahl**: In dieser Teilaufgabe soll eine Schaltung für Schritt 7 entworfen werden. Das Vorzeichen wird auf 0 (positiv) gesetzt, gefolgt von den 8 Exponenten-Bits (`Exp`) und den 23 Fraction-Bits (`Fract`), wodurch eine 32-Bit-Gleitkommazahl (`Sum`) als Output erzeugt wird.

Vervollständige die Schaltung in `6_Assemble.dig`.

g) **Bonusaufgabe**: Füge schlussendlich alle Subcircuits zusammen, sodass ein Gleitkommazahl-Addierer entsteht, der als Input zwei 32-Bit-Gleitkommazahlen (`A` und `B`) erwartet und deren Summe als 32-Bit-Gleitkommazahl (`Sum`) ausgibt.

Vervollständige die Schaltung in `FloatingPointAdder.dig`.

Beachte für die Abgabe auf Artemis:

- Die bereitgestellten Dateinamen, sowie die In- und Outputs dürfen nicht verändert werden.
- Es dürfen ausschließlich Komponenten aus den Kategorien „Logisch“, „Leitungen“, „Multiplexer“ und „Arithmetik“ verwendet werden.
- Für die Bonusaufgabe (Teilaufgabe g) gilt abweichend davon: Es dürfen ausschließlich die Subcircuits der Teilaufgaben a bis f sowie Leitungen verwendet werden. Weitere Komponenten sind für die Bonusaufgabe nicht zugelassen. Einzelne Subcircuits dürfen jedoch auch mehrfach verwendet werden.

5 Referenzmaterial

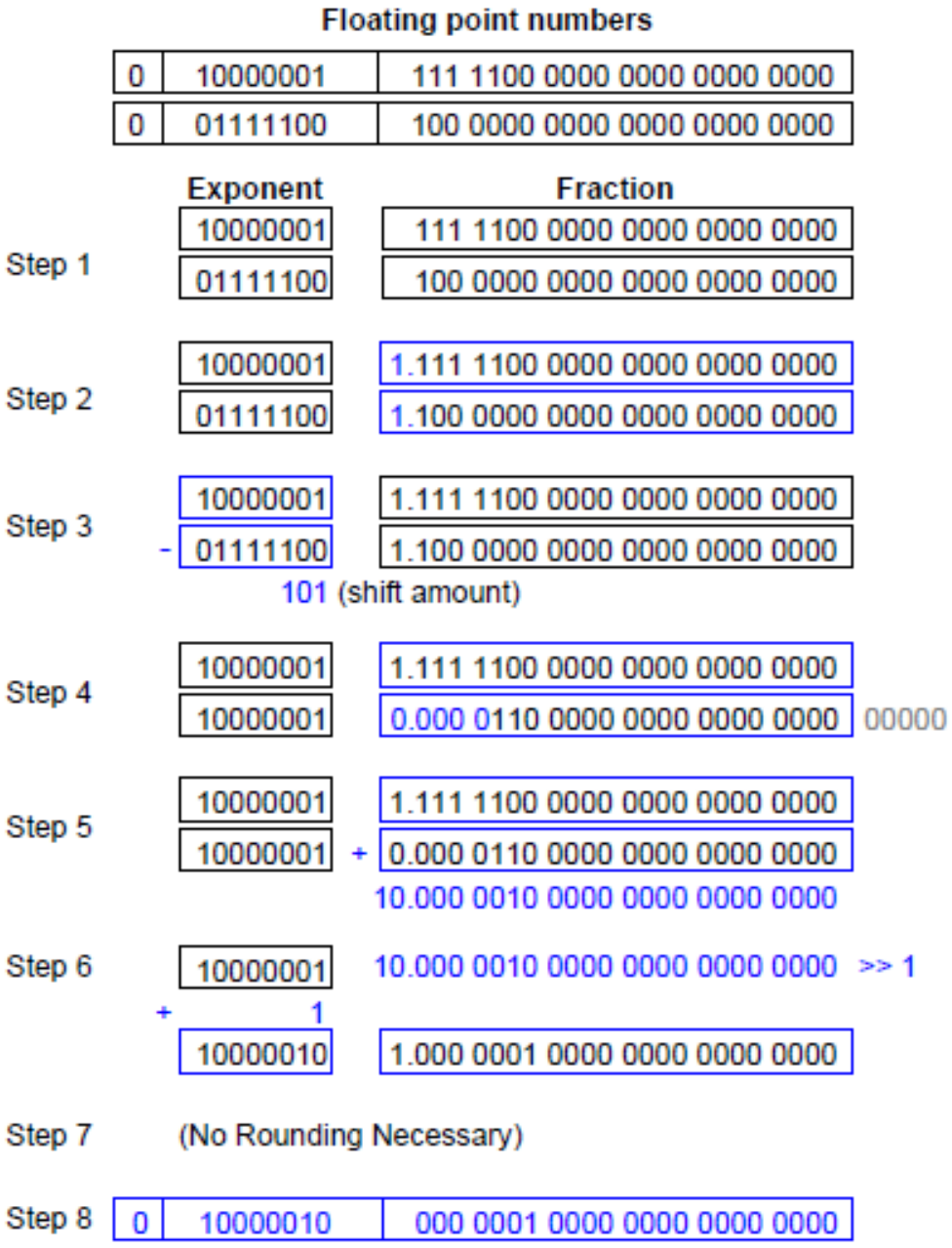


Abbildung 1: Beispiel einer Gleitkomma-Addition

